# Benchmark comparison for glidein factory monitoring between v1_6_beta_1 and v2_0_beta_2 (RRD files and locking)

*Igor Sfiligoi*
*Jan 23rd 2009*

## Introduction

The glidein factory monitoring in v1_X branch is known to be very resource intensive, in particular regarding disk IO operations. The scalability tests performed in Spring 2008 showed that it could not sustain the tested 100 entry points without some kind of resource handling. The solution in the recent v1_X releases has been to implement an internal locking mechanism that reduced the disk IO requests, but also slowed down the responsiveness of the glidein factory.

A different approach is being pursued in the current development branch. The problem of excessive IO operations was assumed to be the large number of RRD files; the v1_X branch puts a single value per RRD file. The development branch instead packs many variables into the RRDs, drastically reducing the number of RRD files (from 131 RRDs per client in v1_X to 9 RRDs per client in the development branch), while preserving the same information.

To validate the above assumptions, I benchmarked v1_6_beta_1 and v2_0_beta_2 cvs tags of glideinWMS. The results are presented below.

## Test setup

I performed all tests on cmssrv13.fnal.gov. This is a 4 Xeon 3.2GHz machine with a single IDE hard drive.

Both glidein factories were configured with 252 entry points (obtained by querying the BDII with minimum requirements). They never ran at the same time.

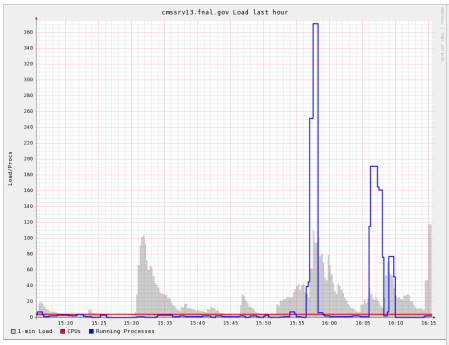
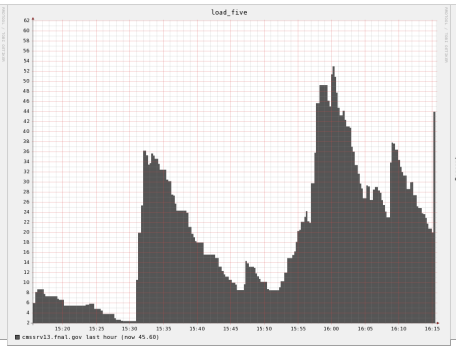
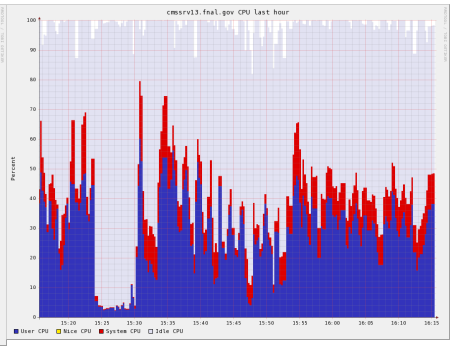
The frontend was configured to always ask for 0 glideins; this is the minimum load setup as it tests the load put in the glidein factory by the monitoring alone.

## Benchmarking results

### *v1_6_beta_1 out of the box*

To establish a baseline, I first tested v1_6_beta_1 as it came from the CVS.

The load on the system appeared reasonable, mostly in the 20s, but the system was responsive and the IO waits low. Below are a few snapshots from the Ganglia monitoring:

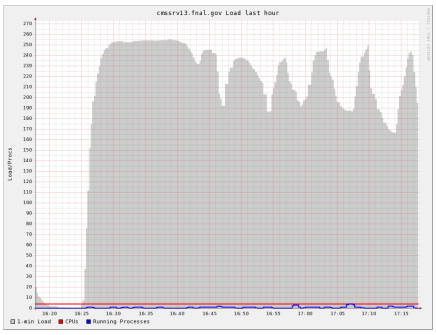
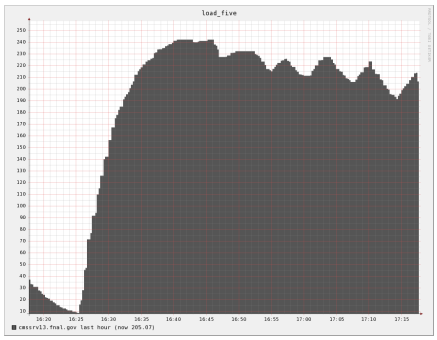
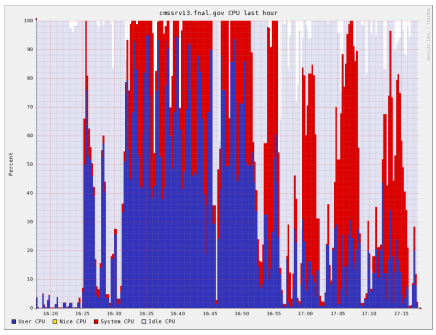
However, the monitoring was taking way too long, slowing the glidein factory to a crawl. Each monitoring update was taking of the order of 20 minutes! A sample log snapshot is shown below:

```
[2009-01-22T15:33:29-05:00 24086] Sleep 60s
[2009-01-22T15:34:29-05:00 24086] Iteration at Thu Jan 22 15:34:29 2009
[2009-01-22T15:34:31-05:00 24086] Client 'cmssrv37_17', requesting 0 glideins

[2009-01-22T15:34:31-05:00 24086]   Params: {u'GLIDEIN_Collector':
u'cmssrv37.fnal.gov:9621,cmssrv37.fnal.gov:9622,cmssrv37.fnal.gov:9623'}

[2009-01-22T15:34:31-05:00 24086]   Decrypted Param Names: []

[2009-01-22T15:34:35-05:00 24086] Client 'cmssrv37_17', schedd status {1: 0},
collector running ?

[2009-01-22T15:34:35-05:00 24086] Writing stats
[2009-01-22T15:51:59-05:00 24086] log_stats written
[2009-01-22T15:53:49-05:00 24086] qc_stats written
[2009-01-22T15:53:49-05:00 24086] Writing lazy stats for qc
[2009-01-22T15:53:49-05:00 24086] Sleep 60s
```

## *v1_6_beta_1 without locking*

Since the system was mostly idle in the previous test, I repeated the test by disabling the locking in the glidein factory code.

The results were quite bad! The system load raised to 250, with the IO wait fraction often in the 90s! The system was essentially unusable during most of the testing period. Below are a few snapshots from the Ganglia monitoring and top:

```
top - 16:45:21 up 198 days,  2:25,  7 users,  load average: 244.24, 241.91, 185.
Tasks: 387 total,   1 running, 386 sleeping,   0 stopped,   0 zombie
Cpu(s):  8.7% us,  2.1% sy,  0.0% ni,  0.0% id, 89.0% wa,  0.2% hi,  0.0% si
Mem:   4148904k total,  3334756k used,   814148k free,   522740k buffers
Swap:  8385920k total,      144k used,  8385776k free,  1398400k cached
```

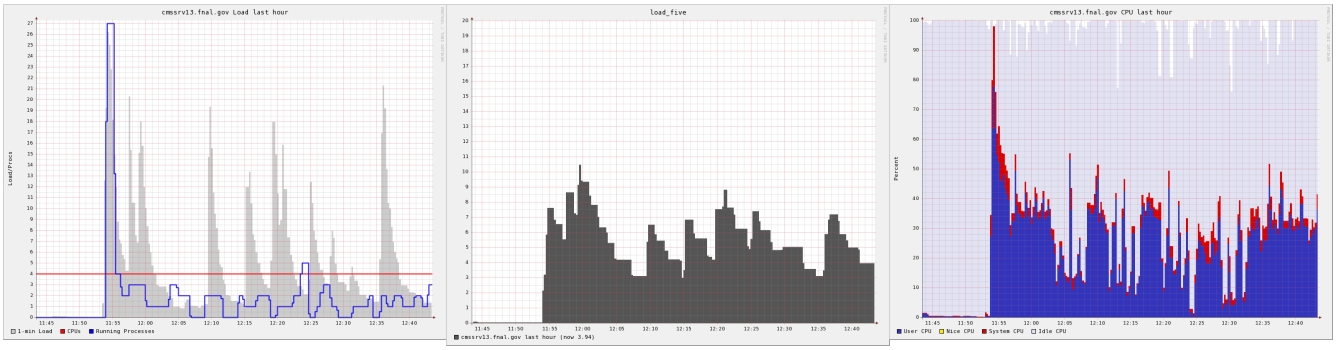
Obviously, with the system essentially trashing, the glidein factory was even slower than before. A sample log snapshot is shown below:

```
[2009-01-22T16:46:04-05:00 5059] Sleep 60s
2009-01-22T16:47:04-05:00 5059] Iteration at Thu Jan 22 16:47:04 2009
[2009-01-22T16:47:07-05:00 5059] Client 'cmssrv37_17', requesting 0 glideins

[2009-01-22T16:47:07-05:00 5059]   Params: {u'GLIDEIN_Collector':
u'cmssrv37.fnal.gov:9621,cmssrv37.fnal.gov:9622,cmssrv37.fnal.gov:9623'}

[2009-01-22T16:47:07-05:00 5059]   Decrypted Param Names: []

[2009-01-22T16:47:13-05:00 5059] Client 'cmssrv37_17', schedd status {1: 0},
collector running ?

[2009-01-22T16:47:13-05:00 5059] Writing stats

[sfiligoi@cmssrv13 log]$ # the cycle did not finish by now
[sfiligoi@cmssrv13 log]$ date
Thu Jan 22 17:14:56 CST 2009
```

## *v2_0_beta_2 out of the box*

The development branch uses less RRD files than v1_X, but the locking is still there. So by just using the code straight from CVS, the locking is still enabled.

Using this setup, the load on the machine was very reasonable, mostly below 10. IO wait readings were also mostly below 1%. Below are a few snapshots from the Ganglia monitoring:



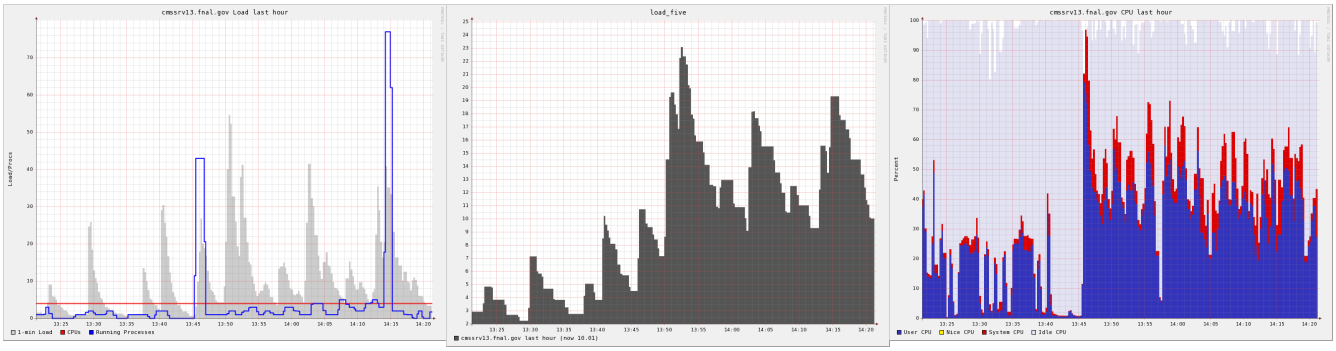

Unfortunately, the glidein factory is very slow here, too. A cycle takes of the order of 30 minutes! A sample log snapshot is shown below:

```
[2009-01-22T13:06:58-05:00 15461] Starting up
[2009-01-22T13:06:58-05:00 15461] Iteration at Thu Jan 22 13:06:58 2009
[2009-01-22T13:07:05-05:00 15461] Client 't10b.test1', requesting 0 glideins
[2009-01-22T13:07:10-05:00 15461] Client 't10b.test1', schedd status {1: 0},
collector running ?
[2009-01-22T13:07:10-05:00 15461] Advertize
[2009-01-22T13:07:12-05:00 15461] Writing stats
[2009-01-22T13:07:12-05:00 15461] log xml written
[2009-01-22T13:36:02-05:00 15461] log_stats written
[2009-01-22T13:36:02-05:00 15461] qc xml written
[2009-01-22T13:38:46-05:00 15461] qc_stats written
[2009-01-22T13:38:46-05:00 15461] Writing lazy stats for logSummary
[2009-01-22T13:38:46-05:00 15461] Writing lazy stats for qc
[2009-01-22T13:38:46-05:00 15461] Sleep 60s
```

## *v2_0_beta_2 without locks*

Since the number of RRDs being written has been substantially reduced, the locking is not really needed anymore. So I removed them from the code and repeated the tests.

Although the load has been increased compared to the previous test, the machine is still very responsive. The IO waits are still below 10% and there are plenty of idle cycles available. Below are a few snapshots from the Ganglia monitoring:




The big difference is in the responsiveness of the glidein factory; a cycle now lasts only one minute, which is more than reasonable. A sample log snapshot is shown below:

```
[2009-01-22T14:17:24-05:00 24114] Sleep 60s
[2009-01-22T14:18:24-05:00 24114] Iteration at Thu Jan 22 14:18:24 2009
[2009-01-22T14:18:29-05:00 24114] Client 't10b.test1', requesting 0 glideins
[2009-01-22T14:18:34-05:00 24114] Client 't10b.test1', schedd status {1: 0},
collector running ?
[2009-01-22T14:18:34-05:00 24114] Writing stats
[2009-01-22T14:18:34-05:00 24114] log xml written
[2009-01-22T14:19:38-05:00 24114] log_stats written
[2009-01-22T14:19:38-05:00 24114] qc xml written
[2009-01-22T14:19:38-05:00 24114] qc_stats written
[2009-01-22T14:19:38-05:00 24114] Sleep 60s
```

## Conclusions

The benchmarking results clearly show that grouping many variables in a single RRD greatly reduces the load on the glidein factory machine, especially in terms of IO load. The direction adopted by the development branch is definitely the right one.

On the other hand, while locking was a good stopgap measure, it does have a significant adverse effect on the responsiveness of the glidein factory. So locking should be avoided whenever feasible (and especially in the new development branch)

## References

glideinWMS home page
http://www.uscms.org/SoftwareComputing/Grid/WMS/glideinWMS/

RRDtool home page
http://oss.oetiker.ch/rrdtool/